Introduction à 

Magalie Fromont

Licence MIASHS / Magistère SME
Universités Rennes 1 et Rennes 2

2016 - 2017

Programme du cours

Références

Références bibliographiques

- Ihaka R. and Gentleman R. (1996). R : a language for data analysis and graphics. *Journal of Computational and Graphical Statistics*.
- Cornillon P.-A. et al. (2012). *Statistiques avec R.*
- Lafaye de Micheaux P. et al. (2014). *Le logiciel R.*

Ressources en ligne

- De (Trop ?) nombreux tutoriels
- Des MOOC
- WikiStat : `http://www.math.univ-toulouse.fr/~besse/Wikistat`
- Une référence officielle : `http://www.r-project.org/`

Programme

L'environnement R - Présentation

R est un système ou un environnement :

- créé dans les années 1990 par R. Ihaka et R. Gentleman (Université d'Auckland, Nouvelle Zélande), développé depuis 1997 par la R Development Core Team,
- dédié à l'analyse de données et le traitement graphique,
- basé sur le langage de programmation S (R. Becker, J. Chambers et A. Wilks, Bell laboratories),
- diffusé librement selon le principe de licence GNU,
- disponible sous de nombreux systèmes d'exploitation (UNIX, Windows, MacOS),
- disposant d'interfaces avec de nombreux autres langages ou logiciels (C, Java, Spad, SAS, Excel, Latex...).

L'environnement R - Installation

L'installation de R se fait à partir du site référence

`http://www.r-project.org/`

- Dans le menu `Download`, cliquer sur `CRAN`.
- Choisir un site miroir du CRAN (Comprehensive R Archive Network) proche de chez soi.
- Dans `Download and Install R`, choisir son système d'exploitation.
- Télécharger le ou les fichiers d'installation (base, contrib, tools) et exécuter ce ou ces fichiers.

L'environnement R - GUIs et IDEs

Interfaces graphiques (GUIs ou Graphical User Interfaces)

RGUI (par défaut sous Windows), R console (par défaut sous MacOS), RCommander, JavaGUI for R, Rattle...

Éditeurs et Environnements de développement intégrés (IDEs ou Integrated Development Environments)

Tinn-R (Windows), Eclipse, RStudio...

RGUI, interface graphique par défaut sous Windows

- Plusieurs fenêtres : fenêtre principale (la console), fenêtres graphiques, fenêtres d'informations
- Menu `File` : gestion de l'espace de travail
- Menu `Edit` : commandes usuelles de copier-coller, personnalisation de l'interface
- Menu `View` : affichage ou non des barres d'outils et de statut
- Menu `Misc` : gestion des objets en mémoire, calculs en cours
- Menu `Packages` : gestion des packages
- Autres menus usuels des applications Windows

RStudio, Environnement de développement intégré multi-plateforme

- Une fenêtre principale (la console)
- Une fenêtre éditeur de texte avec syntaxes intégrées, permettant l'exécution directe de codes
- Une fenêtre comprenant divers outils : gestion des fichiers, des graphiques, des packages, aide, visualisation, historique
- Menus `File`, `Edit`, `Code`, `View`...
- Menu `Session` : gestion de l'espace de travail

L'environnement R - Fonctions et packages

Le logiciel R comprend des programmes sous forme de fonctions et des jeux de données, organisés en packages (ou bibliothèques).

Les packages disponibles sont extrêmement nombreux. Des mises à jour de ces packages et de nouveaux packages voient le jour régulièrement.

Lors de l'installation de R, des packages de base sont mis à disposition de l'utilisateur. Les packages additionnels nécessaires devront, eux, être installés puis chargés à chaque nouvelle session de travail.

Programme

2. Premiers pas
   - Gestion d'une session de travail
   - Exécution d'un script
   - Gestion des packages
   - Utilisation des aides

Premiers pas - Gestion d'une session de travail

Ouvrir une session de travail

Cliquer sur l'icone RStudio : une session de travail s'ouvre.

Dans la console, le prompt $>$ s'affiche à la fin du message de démarrage, indiquant que R attend une instruction.

Une instruction est exécutée en appuyant sur la touche `Entrée`.

Si l'instruction est complète, le prompt s'affiche à nouveau.
Si l'instruction est incomplète, le prompt est remplacé par $+ \rightarrow$ compléter la ligne ou cliquer sur `STOP` ou appuyer sur `Echap`.
Si l'instruction est erronée, un message d'erreur apparaît.

Rappel d'instructions

Les flèches au clavier permettent de rappeler des instructions.
`history(10)` permet de récupérer l'historique des 10 dernières instructions.
On peut aussi retrouver l'historique complète depuis l'outil `History`.

Caractères spéciaux

;	Permet de séparer des instructions sur une même ligne ou d'exécuter une instruction sans en afficher le résultat.
♯	Permet de mettre des commentaires. La ligne qui suit ce caractère n'est pas comprise comme une instruction.

Quitter une session, sauvegarder, charger l'espace de travail

Pour quitter une session R :

- depuis la console : `q()`, ou,
- fermer l'application RStudio de façon classique.

`Sauver une image de la session ?` [`oui/non/annuler`] s'affiche → Sauvegarder ou non l'image de la session (ou espace de travail) dans des fichiers cachés `.RData` et `.Rhistory` du répertoire courant.

Pour charger dans une nouvelle session les objets contenus dans `.RData` et/ou l'historique contenue dans `.Rhistory`,

- depuis la console : `load(".RData")`/`loadhistory(".Rhistory")`,
- depuis l'outil `Files` : en cliquant sur `.RData`/`.Rhistory`.

On peut aussi utiliser `save.image ("../travail.RData")` pour sauvegarder les objets de la session dans un fichier de son choix, puis `load ("../travail.RData")` pour charger ces objets dans une nouvelle session. Attention : les chemins dans R sont écrits à l'aide de / et non \ !

Gérer le répertoire de travail depuis la console

`getwd()`	Connaître le répertoire courant.
`setwd("../FormationR")`	Choisir `FormationR` comme répertoire courant.
`list.files()`	Afficher la liste des fichiers du répertoire.

↪ Le choix du répertoire courant peut se faire depuis le Menu `Session`.

Premiers pas - Exécution d'un script

Gérer un script

Un nouveau script peut être créé et enregistré sous `nom.script.R` dans le répertoire courant choisi, depuis le Menu `File` → `New File`.

Un ancien script peut être récupéré depuis le Menu `File` → `Open File`, ou en cliquant sur l'onglet correspondant s'il apparaît.

Le répertoire dans lequel est enregistré le script peut alors être choisi comme répertoire courant depuis le Menu `Session` → `Set Working Directory` → `To Source File Location`.

Exécuter un script

Pour exécuter les instructions d'un script, on peut :

- procéder par copier-coller dans la console
- exécuter toutes les instructions du script avec `source(file="../nom.script.R")`
- surligner les instructions du script à exécuter et cliquer sur `Run`
- se positionner sur la ligne à exécuter et cliquer sur `Run`

Rediriger les sorties

On peut rediriger les sorties R dans un fichier texte avec `sink(file="../sorties.txt")`, et stopper la redirection avec `sink()`.

Premiers pas - Gestion des packages

Gérer les packages depuis la console

`update.packages()`	Permet de mettre à jour un package.
`install.packages (dependencies=TRUE)`	Permet d'installer un package additionnel : choisir un site miroir du CRAN et le package à installer.
`install.packages (repos=NULL,pkgs= "../nom.fichier")`	Permet d'installer un package additionnel à partir d'un fichier téléchargé sur le CRAN au préalable.
`library(nom.package)`	Appeler un package additionnel installé préalablement. Instruction à donner à chaque nouvelle session de R !
`library()`	Afficher la liste des packages installés.

Gestion des packages depuis l'outil `Packages`

Pour mettre à jour des packages : choisir les packages à mettre à jour et cliquer sur `Update`.

Pour installer des packages additionnels : cliquer sur `Install`.
Deux voies sont alors possibles :

- Dans `Install from` : choisir `Repository (CRAN)` puis indiquer le package à installer.
- Dans `Install from` : choisir `Package Archive File` puis choisir le fichier archive téléchargé sur le site du CRAN au préalable.

Rappel : ces packages devront être appelés à chaque nouvelle session de R avec `library(nom.package)` !

Premiers pas - Utilisation des aides

Obtenir de l'aide depuis la console

`help.start()`	Ouvrir l'aide en ligne. Parcourir les Packages est souvent utile.
`help(mean)` ou `?mean`	Afficher l'aide sur la fonction `mean`.
`help(package=base)`	Afficher l'aide sur le package `base`.
`help.search(mean)`	Rechercher la fonction `mean` dans les packages R.
`RSiteSearch("calcul moyenne")`	Chercher de l'aide dans les archives des utilisateurs de R.

↪ Les aides peuvent s'obtenir depuis l'outil `Help`

L'aide sur une fonction donne :

- Sa description succinte
- La liste des arguments à renseigner en entrée, avec éventuellement leurs valeurs par défaut
- La liste des arguments en sortie
- Le nom du (des) programmateur(s)
- Un ou plusieurs exemples d'utilisation de la fonction qui peuvent être copiés-collés dans la console pour comprendre le fonctionnement de la fonction ou exécutés directement avec la commande `example(mean)`.

Programme

3. Les objets
   - Généralités
   - Les vecteurs
   - Les matrices
   - Les facteurs
   - Les listes
   - Les data.frames
   - Les data.tables
   - Les Dates

Les objets - Généralités

Créer, afficher, sauvegarder des objets

On peut créer un objet `x` :

- par affectation avec : `x<-`, `->x` ou `x=`.
- en chargeant un fichier contenant `x` : `load("../travail.RData")`.

`print(x)` ou `x`	Afficher l'objet `x`.
`View(x)`	Afficher l'objet `x` sous forme de table.
`summary(x)`	Afficher un résumé de `x`.
`head(x)/tail(x)`	Afficher les premiers/derniers éléments de `x`.
`save(x,file="../travail.RData")`	Sauvegarder l'objet `x` dans le fichier `travail.RData`.
`objects()/ls()/ls.str()`	Afficher la liste des objets en mémoire.
`ls(pat="a")/ls(pat="^a")`	Afficher la liste des objets dont le nom contient/commence par le caractère "a".

Effacer des objets

`rm(x)`	Effacer l'objet `x` de la mémoire.
`rm(x,y)`	Effacer les objets `x` et `y` de la mémoire.
`rm(list=ls())`	Effacer tous les objets de la mémoire.
`rm(list=ls(pat="a"))`	Effacer tous les objets dont le nom contient le caractère "a".
`rm(list=ls(pat="^a"))`	Effacer tous les objets dont le nom commence par "a".

Les objets peuvent être de différentes classes :

- des vecteurs,
- des matrices,
- des facteurs,
- des listes,
- des tableaux de données ou data.frames,
- d'autres plus récentes (data.table, Date, ts...)

Connaître la classe des objets

`class(x)`	Afficher la classe de l'objet `x` (`vector`, `matrix`, `factor`, `list`, `data.frame`...).
`is.vector(x)`	Tester si l'objet `x` est de classe `vector`.
`is.matrix(x)`	Tester si l'objet `x` est de classe `matrix`.
`is.factor(x)`	Tester si l'objet `x` est de classe `factor`.
`is.list(x)`	Tester si l'objet `x` est de classe `list`.
`is.data.frame(x)`	Tester si l'objet `x` est de classe `data.frame`.
`is.data.table(x)`	Tester si l'objet `x` est de classe `data.table`.

Convertir la classe des objets

`as.vector(x)`	Convertir la classe de l'objet `x` en `vector`.
`as.matrix(x)`	Convertir la classe de l'objet `x` en `matrix`.
`as.factor(x)`	Convertir la classe de l'objet `x` en `factor`.
`as.list(x)`	Convertir la classe de l'objet `x` en `list`.
`as.data.frame(x)`	Convertir la classe de l'objet `x` en `data.frame`.

Les objets ont, selon leur classe, des attributs intrinsèques : le mode, la longueur, et éventuellement d'autres.

Connaître les attributs des objets

`mode(x)`	Afficher le mode de l'objet `x` (`numeric`, `character`, `logical`, `complex`, `null`...).
`length(x)`	Afficher la longueur de l'objet `x`.
`attributes(x)`	Afficher les éventuels attributs spécifiques.

Connaître le mode des objets (suite)

`is.numeric(x)`	Tester si l'objet `x` est de mode `numeric`.
`is.character(x)`	Tester si l'objet `x` est de mode `character`.
`is.logical(x)`	Tester si l'objet `x` est de mode `logical`.
`is.complex(x)`	Tester si l'objet `x` est de mode `complex`.
`is.null(x)`	Tester si l'objet `x` est de mode `null`.

Convertir le mode ou autre attribut des objets

`as.numeric(x)`	Convertir le mode de l'objet `x` en `numeric`.
`as.character(x)`	Convertir le mode de l'objet `x` en `character`.
`as.logical(x)`	Convertir le mode de l'objet `x` en `logical`.
`as.complex(x)`	Convertir le mode de l'objet `x` en `complex`.
`attr(x,"dim")=`	Modifier l'attribut `dim` de l'objet `x`.

Les valeurs manquantes dans les objets

Elles sont notées `NA` (not available) ou NaN (not a number).

`is.na(x)`	Savoir si l'objet `x` contient des valeurs manquantes.
`is.nan(x)`	Ces instructions retournent un objet de mode `logical` de même longueur que `x`.
`na.fail(x)`	Retourner un message d'erreur si l'objet `x` contient au moins une valeur manquante.

Les objets - Les vecteurs

Un vecteur est une combinaison d'éléments de même mode.

Créer un vecteur numérique

`x=2, x=pi, x=Inf, x=2e-1`	Créer un vecteur numérique de taille 1 contenant le nombre 2, π, ∞, ou 0.2...
`c(1,1,2,3,5,8)`	Créer un vecteur contenant 1,1,2,3,5,8.
`scan(n=3)`	Créer un vecteur de taille 3, en entrant ses éléments au fur et à mesure que R le demande.
`scan()`	Créer un vecteur en entrant ses éléments au fur et à mesure, et terminer avec Entrée.
`1:10/sequence(10)`	Créer le vecteur des entiers de 1 à 10.
`seq(1,5,by=0.5)`	Créer un vecteur contenant une séquence de réels
`seq(1,5,length=4)`	allant de 1 à 5, de pas 0.5 ou de longueur 4.
`sequence(c(7,5))`	Créer le vecteur des entiers de 1 à 7, puis de 1 à 5.
`numeric(5)`	Créer un vecteur contenant 5 fois 0.
`rep(1,5)`	Créer un vecteur contenant 5 fois le réel 1.

Créer un vecteur numérique aléatoire

On peut générer des vecteurs contenant des simulations de réalisations d'un n échantillon d'une loi réelle donnée à l'aide de `rloi(n,`p_1`,`p_2`,...)`, où `loi` indique la loi de probabilité, $p_1, p_2, \ldots$ les paramètres de la loi de probabilité.

Exemples : `rnorm(n,0,1)`, `rexp(n,1)`, `rpois(n,2)`, `rchisq(n,100)`, `rt(n,100)`, `rf(n,100,50)`, `rbinom(n,100,0.5)`, `runif(n,0,1)`...

`set.seed(k)` permet de choisir une graine pour le générateur de nombres pseudo-aléatoires.

Les opérateurs et les fonctions numériques agissent sur les vecteurs élément par élément.

Les opérateurs numériques

+ (addition), - (soustraction), * (multiplication), / (division), ^ (puissance), %% (congruence), %/% (division entière).

Les fonctions numériques

Les fonctions numériques de base sont toutes disponibles dans R (`sqrt`, `abs`, `log`, `exp`, `log10`, `sin`, `cos`, `tan`, `asin`, ...), ainsi que les fonctions issues de lois de probabilités (`dloi`, `ploi` et `qloi` correspondent à la densité, la fonction de répartition, la fonction quantile de la loi de probabilité `loi`).

Les fonctions numériques (suite)

`sum(x)`	Calculer la somme des éléments de `x`.
`prod(x)`	Calculer le produit des éléments de `x`.
`diff(x)`	Calculer la différence entre les éléments consécutifs de `x`.
`max(x)`	Donner le maximum des éléments de `x`.
`min(x)`	Donner le minimum des éléments de `x`.
`range(x)`	Donner les minimum et maximum de `x`.
`which.max(x)`	Donner l'indice du maximum de `x`.
`which.min(x)`	Donner l'indice du minimum de `x`.
`sign(x)`	Donner le signe de `x`.
`ceiling(x)`	Donner l'entier supérieur ou égal à `x`.
`floor(x)`	Donner la partie entière de `x`.
`round(x,2)`	Arrondir les éléments de `x` à 2 décimales.
`signif(x,2)`	Arrondir les éléments de `x` avec 2 chiffres.

Les fonctions numériques (suite)

`sort(x)`	Ordonner les éléments de `x`.
`rev(sort(x))`	Ordonner les éléments de `x` dans l'ordre décroissant.
`rank(x)`	Donner les rangs des éléments de `x`.
`order(x)`	Donner l'indice du plus petit élément de `x`, du deuxième plus petit élément de `x`, etc.
`pmin(x,y)`/ `pmax(x,y)`	Donner un vecteur dont les éléments sont les minima/maxima entre les éléments de `x` et `y`.
`cumsum(x)`/ `cumprod(x)`	Donner un vecteur dont le ième élément est la somme/produit des i premiers éléments de `x`.
`cummin(x)`/ `cummax(x)`	Donner un vecteur dont le ième élément est le minimum/maximum des i premiers éléments de `x`.
`choose(n,k)`	Calculer la combinaison de k éléments parmi n.

Les fonctions statistiques de base

`summary(x)`	Donner un résumé statistique de `x`.
`mean(x)`	Calculer la moyenne empirique des éléments de `x`.
`var(x)/cov(x)`	Calculer la variance empirique de `x`.
`sd(x)`	Calculer l'écart type empirique de `x`.
`scale(x)`	Centrer et réduire empiriquement `x`.
`median(x)`	Calculer la médiane empirique de `x`.
`var(x,y)/cov(x,y)`	Calculer la covariance empirique entre les éléments de `x` et ceux de `y`.
`corr(x,y)`	Calculer la corrélation linéaire entre `x` et `y`.
`quantile(x)`	Calculer le quantile empirique de `x`.

Créer un vecteur de caractères

`x="a"` ou `x='a'`	Créer un vecteur contenant le caractère "a".
`c("a","b","c")`	Créer un vecteur contenant "a","b","c".
`LETTERS, letters`	Créer le vecteur des lettres de l'alphabet.
`month.abb, month.name`	Créer le vecteur des mois de l'année.
`format(1:10)`	Créer un vecteur contenant "1",...,"10".
`character(5)`	Créer un vecteur contenant 5 fois "".
`rep("a",5)`	Créer un vecteur contenant 5 fois "a".
`paste("A",1:5,sep="")`	Créer un vecteur contenant "A1",...,"A5".
`paste("A",1:5,sep="-")`	Créer un vecteur contenant "A-1",...,"A-5".
`paste(c("A","B"),1:2, txt,sep=".")`	Créer un vecteur contenant "A.1.txt","B.2.txt"
`paste("A",1:3,sep="", collapse="+")`	Créer le vecteur "A1+A2+A3".
`x="Il dit\"blabla\""`	Créer un vecteur contenant des guillemets.

Manipuler un vecteur de caractères

`substr("blabla",4,6)`	Extraire ou remplacer le caractère formé des lettres 4 à 6 de "blabla".
`grep("bla",x)`	Donner les indices des éléments de `x` contenant "bla".
`grep("[a-c]",x)`	Donner les indices des éléments de `x` contenant "a", "b" ou "c".
`sub("bla","pa",x)`	Remplacer le premier "bla" de `x` par "pa".
`gsub("bla","pa",x)`	Remplacer tous les "bla" de `x` par "pa".
`tolower("BLABLA")`	Ecrire "blabla" en minuscules.
`toupper("blabla")`	Ecrire "BLABLA" en majuscules.

Créer un vecteur logique

`x=TRUE`/`x=FALSE`	Créer un vecteur logique de taille 1 contenant `TRUE`/ `FALSE`.
`c(TRUE,FALSE)`	Créer un vecteur logique contenant `TRUE` et `FALSE`.
`logical(5)`	Créer un vecteur logique de taille 5 ne contenant que des `FALSE`.

Les vecteurs logiques les plus utiles sont obtenus à partir des opérateurs de comparaison (`!=`,`==`,`<`,`>`,`<=`,`>=`).
%in% permet par ailleurs de tester l'appartenance.

Exemples : `1<=2`, `pi==3.14`, `x>0 & x<1`, `1%in%1:3`.

Les opérateurs logiques

Les opérateurs logiques sont : `!x` (non logique), `x&y` (et logique opérant sur tous les éléments de x et y), `x&&y` (et logique opérant sur le premier élément de x et y), `x|y` (ou logique opérant sur tous les éléments de x et y), `x||y` (ou logique opérant sur le premier élément), `xor(x,y)` (ou exclusif).

On peut utiliser les opérateurs numériques, les `FALSE` étant transformés en 0, les `TRUE` en 1.

Les fonctions logiques

`all(x)`	Tester si tous les éléments de `x` sont `TRUE`.
`any(x)`	Tester si au moins un des éléments de `x` est `TRUE`.

Manipuler des vecteurs : extraire des éléments

L'extraction d'éléments d'un vecteur `x` s'opère à l'aide de `x[extract]`, où `extract` est soit :

- un vecteur d'entiers positifs ou négatifs,
- un vecteur logique.

`x[3]`	Extraire le 3ème élément de `x`.
`x[c(2,3)]`	Extraire les 2ème et 3ème éléments de `x`.
`x[-c(2,3)]`	Extraire tous les éléments de `x` sauf les 2 et 3èmes.
`x[x>=0]`	Extraire les valeurs positives de `x`.
`x[x%%2==0]`	Extraire les valeurs paires de `x`.
`x[!is.na(x)]`	Extraire les valeurs non manquantes de `x`.
`na.omit(x)`	
`x[is.na(x)]=0`	Remplacer les valeurs manquantes de `x` par 0.

Manipuler des vecteurs : fonctions générales

`x=c(y,z)`	Concaténer les vecteurs `x` et `y`.
`rev(x)`	Inverser l'ordre des éléments de `x`.
`match(x,y)`	Donner un vecteur de même longueur que `x` contenant les éléments de `x` qui sont dans `y` (`NA` sinon).
`which(x==2)`	Donner les indices des éléments de `x` égaux à 2.
`unique(x)`	Supprimer les éléments dupliqués de `x`.
`table(x)`	Créer un tableau des effectifs des éléments de `x`.
`sample(x,3)`	Tirer aléatoirement sans remise 3 éléments dans `x`.
`sample(x,3, replace=TRUE)`	Tirer aléatoirement avec remise 3 éléments dans `x`.

Les objets - Les matrices

Une matrice est composée d'éléments de même mode.
Elle possède un attribut spécifique : `dim`.

Créer des matrices à partir de vecteurs

`matrix(1:6,2,3)`	Créer une matrice 2×3 remplie colonne par
`x=1:6;dim(x)=c(2,3)`	colonne par les éléments du vecteur `1:6`.
`matrix(1:6,2,3,byrow=T)`	Créer une matrice 2×3 remplie ligne par ligne par les éléments de `1:6`.
`rbind(vect1,vect2)`	Créer une matrice en concaténant des vecteurs par colonne.
`cbind(vect1,vect2)`	Créer une matrice en concaténant des vecteurs par colonne.

Créer des matrices particulières

`matrix(1,nr=2,nc=2)`	Créer une matrice 2×2 ne contenant que des 1.
`diag(3)`	Créer la matrice identité 3×3.
`diag(c(1,2,3))`	Créer une matrice diagonale d'éléments diagonaux 1,2,3.
`diag(c(1,2,3),nr=3,nc=5)`	Créer une matrice 3×5 dont la première sous-matrice 3×3 est diag(c(1,2,3)).

Les opérateurs et fonctions vectoriels

Les opérateurs vectoriels sont utilisables pour des matrices de même dimension, agissant élément par élément, comme certaines fonctions vectorielles.
On peut appliquer une fonction à des éléments choisis d'une matrice à l'aide de `apply()` :

`apply(x,1,fun)`/	Appliquer `fun` aux éléments de `x` ligne
`apply(x,2,fun)`	par ligne/colonne par colonne.
`apply(x,c(1,2),fun)`	Appliquer `fun` aux éléments de `x` ligne par ligne et colonne par colonne.

Les opérateurs et fonctions matriciels

`nrow(x)/ncol(x)`	Donner le nombre de lignes/colonnes de `x`.
`NROW(x)/NCOL(x)`	Idem mais valables pour les vecteurs aussi.
`t(x)`	Calculer la transposée de `x`.
`x%*%y`	Calculer le produit matriciel de `x` par `y`.
`det(x)`	Calculer le déterminant de `x` (matrice carrée).
`solve(x)`	Inverser `x`.
`solve(A,b)`	Résoudre l'équation `Ax=b`.
`e=eigen(x)`	Calculer les valeurs et vecteurs propres de `x`.
`e$val`	Donner les valeurs propres de `x`.
`e$vec`	Donner les vecteurs propres de `x`.
`svd(x)`	Donner la décomposition en valeurs singulières de `x`.
`chol(x)`	Donner la décomposition de Cholesky de `x`.
`qr(x)`	Donner la décomposition QR de `x`.

Manipuler des matrices

`rbind(x,y)`	Juxtaposer les matrices `x` et `y` (côte à côte ou l'une
`cbind(x,y)`	au dessous de l'autre).
`diag(x)=0`	Remplacer les éléments diagonaux de `x` par 0.
`diag(x)`	Extraire la diagonale de `x`.
`x[i,j]`	Extraire l'élément (i,j) de `x`.
`x[i,]`	Extraire la ligne i de `x`.
`x[,j]`	Extraire la colonne j de `x`.
`x[-c(1,5),]`	Extraire tous les éléments de `x` sauf les lignes 1 et 5.

L'attribut optionnel `dimnames`

`dimnames(x)=`	Attribuer des noms aux lignes et aux
`list(nomsl,nomsc)`	colonnes de la matrice `x`.
`dimnames(x)=`	Attribuer des noms aux colonnes de la
`list(NULL,nomsc)`	matrice `x`.
`dimnames(x)=list(NULL,`	Supprimer les noms des lignes.
`dimnames(x)[[2]])`	

↪ Utile pour extraire des éléments de `x`.

Les objets - Les facteurs

Un facteur est constitué des valeurs d'une variable catégorielle et des niveaux possibles de cette variable (même non visibles).
Attribut spécifique : `levels`.

Créer des facteurs

`factor(1:3)`	Créer un facteur de valeurs 1 à 3, de niveaux 1 à 3.
`factor(1:3,levels=1:5)`	Créer un facteur de valeurs 1 à 3, de niveaux 1 à 5.
`factor(c("a","b"))`	Créer un facteur de valeurs "a", "b", de niveaux "a" et "b".
`factor(1:2,labels=c("a","b"))`	Idem.
`factor(1:3, exclude=2)`	Créer un facteur de valeurs 1,NA,3 de niveaux 1,3.

Créer des facteurs (suite)

`gl(2,3)`	Créer un facteur de valeurs 1,1,1,2,2,2 et de niveaux 1,2.
`gl(2,3,length=30)`	Créer un facteur de valeurs 1,1,1,2,2,2, répétées 5 fois (longueur totale=30) de niveaux 1,2.
`gl(2,3, labels=c("a","b"))`	Créer un facteur de valeurs "a","a","a", "b","b","b" de niveaux "a","b".
`cut(x,5)`	Créer un facteur à 5 niveaux à partir d'un vecteur numérique `x` (découpe en classes).

Manipuler des facteurs

`nlevels(x)`	Donner le nombre de niveaux du facteur `x`.
`ordered(x)`	Transformer `x` en facteur ordonné selon les niveaux.

Utiliser des facteurs

On considère une population, dont le sexe, l'âge et la taille de chaque individu sont donnés par un facteur `sexe` contenant les valeurs "M","F" de niveaux "M","F", un facteur age et un vecteur numérique `taille`.

Pour appliquer une fonction `fun` aux éléments de `taille` selon les niveaux de `sexe` : `tapply(taille,sexe,fun)`, selon ceux de `sexe` et de age : `taggregate(taille,list(sexe,age),fun)`.

Les objets - Les listes

Une liste est un ensemble ordonné d'objets (composantes) dénommés ou non, pas nécessairement de même mode et de même longueur.
Attribut spécifique : `names`.

Créer des listes à partir d'objets

`maliste=list(x,y,z)`	Créer une liste de composantes `x,y,z`.

Manipuler des listes

`names(maliste)`	Afficher les noms des composantes de `maliste`.
`names(maliste)=...`	Donner des noms aux composantes de `maliste`.
`maliste[[3]]`	Extraire le 3ème objet de `maliste`.
`maliste$objet`	Extraire l'objet dénommé `objet` de `maliste`.
`maliste[["objet"]]`	Idem.
`maliste[c(1,3)]`	Extraire les 1er et 3ème objets de `maliste`.
`c(liste1,liste2)`	Concaténer `liste1` et `liste2`.
`lapply(maliste,fun)`	Appliquer `fun` à chacun des objets de `maliste`.
`unlist(maliste)`	Mettre les objets de `maliste` dans un vecteur.

Les objets - Les data.frames

Les data.frames sont des listes dont les composantes sont de même longueur : format individus $\times$ variables $\rightarrow$ objets fondamentaux de l'analyse statistique.

Créer des data.frames

L'importation de données dans un package R à l'aide de `data()` ou dans un fichier externe à l'aide de `read.table()` crée un data.frame.

La commande `data.frame(x,y,...)` permet de créer un data.frame dont les composantes sont les objets `x,y...` (de même longueur).

Exemple

```
> x=factor(c("M","F","F","M","F","F"))
> y=c(1.75,1.68,1.72,1.67,1.59,1.65)
> data1=data.frame(y,x)
> noms=paste("Individu",1:6,sep=" ")
> data2=data.frame(taille=y,sexe=x,row.names=noms)
```

Manipuler des data.frames

`summary(data2)`	Afficher un résumé de `data2`.
`names(data2)`	Donner les noms des composantes de `data2`.
`cbind(data1,data2)`	Concaténer `data1` et `data2` par colonnes.
`rbind(data1,data2)`	Concaténer `data1` et `data2` par lignes.
`merge(data1,data2, by="key")`	Concaténer `data1` et `data2` suivant la clef `key`.
`data2[[1]]`	Extraire le 1er objet de `data2`.
`data2$taille`	Extraire l'objet `taille` de `data2`.
`data2[["taille"]]`	Idem.
`data2[,c(2,3)]`	Extraire les 2ème et 3ème objets de `data2`.
`data2[2,3]`	Extraire le 2ème élt du 3ème objet de `data2`.
`data2[1:10,]`	Extraire les 10 premières lignes de `data2`.
`head(data2,n=10L)`	Idem.
`data2[data2$sexe=="F",]`	Extraire les lignes de `data2` pour lesquelles `sexe` est égal à `"F"`.
`split(data2,data2$sexe)`	Séparer `data2` en plusieurs data.frames suivant les modalités de `sexe`, et les mettre dans une liste.

Manipuler des data.frames (suite)

`lapply(data2,fun)`	Appliquer `fun` aux composantes de `data2`.
`apply(data2,2,fun)`	Appliquer `fun` aux colonnes de `data2`.
`tapply(data2,fact, fun)`	Appliquer `fun` aux élts de `data2` précisés dans `fact` (facteur ou liste de facteurs).
`aggregate(data2,fact, fun)`	Appliquer `fun` aux éléments de `data2` précisés dans une liste d'éléments de même taille que les objets de `data2`.
`by(data2,fact,fun)`	Appliquer `fun` aux éléments de `data2` selon les niveaux de `fact` (facteur ou liste).

Attacher des data.frames (à éviter !)

`attach(data2)`	Ajouter `data2` aux répertoires de travail en position 2. Enregistrer dans ce répertoire les objets `taille` et `sexe` de `data2`.
`taille`	Donner les valeurs de `data2$taille`.
`taille[sexe=="M"]`	Extraire les éléments de `taille` correspondant au niveau "M" du facteur `sexe`.
`detach(data2)`	Détacher le répertoire `data2`.

Les objets - Les data.tables

Les objets récents prévus pour l'extraction, la programmation et l'agrégation rapides : les data.tables !
↪ Les data.tables héritent des data.frames donc les fonctions applicables aux data.frames sont applicables aux data.tables.

Créer des data.tables

`library(data.table)`	Appeler le package `data.table`.
`data.t=data.table (taille=y,sexe=x)`	Créer un data.table de la même façon qu'un data.frame
`data.t=data.table(data.f)`	Transformer le data.frame `data.f` en data.table.

Afficher des data.tables

`tables()`	Afficher la liste des data.tables en mémoire dans un data.table.
`sapply(data.t,class)`	Afficher les classes des objets de `data.t`.

Manipuler des data.tables

`data.t[2]`	Extraire la 2ème ligne de `data.t`.
`data.t[!2]`	Extraire toutes les lignes sauf la 2ème.
`data.t[,taille]`	Extraire l'objet `taille` de `data.t` sous forme de vecteur.
`data.t[,list(taille)]`	Extraire l'objet `taille` de `data.t` sous forme de data.table.
`data.t[sexe=="F"]`	Extraire les lignes de `data.t` pour lesquelles `sexe` est égal à `"F"`.

Manipuler des data.tables avec une clef

`setkey(data.t,sexe)`	Définir comme clef l'objet `sexe`. Les lignes seront rangées par ordre croissant selon les valeurs de `sexe`.
`haskey(data.t)`	Vérifier si `data.t` a une clef.
`key(data.t)`	Donner la clef de `data.t`.
`data.t["F"]`	Extraire les lignes de `data.t` pour lesquelles la clef (ici `sexe`) est égale à `"F"`.

Programmation rapide à l'aide de data.tables

`data.t[,fun(taille)]`	Appliquer `fun` à l'objet `taille`.
`data.t[1:2,fun(taille)]`	Appliquer `fun` aux 2 premiers éléments de l'objet `taille`.
`data.t[,fun(taille), by=sexe]`	Appliquer `fun` à l'objet `taille` selon les valeurs du facteur `sexe`.
`data.t["F",fun(taille)]`	Appliquer `fun` aux éléments de l'objet `taille` correspondant au `sexe` `"F"`.
`data.t[c("F","M"), fun(taille)]`	Appliquer `fun` aux éléments de `taille` correspondant au `sexe` `"F"` ou `"M"`.
`data.t[c("F","M"), fun(taille),by=.EACHI]`	Appliquer `fun` aux éléments de `taille` correspondant au `sexe` `"F"` puis `"M"`.

Agrégation rapide de data.tables

`X[Y]`	Agréger les deux data.tables `X` et `Y`.
`X[Y,nomatch=0]`	Agréger `X` et `Y` sans les données manquantes.
`X[Y,fun(V),by=.EACHI]`	Agréger, calculer `fun(V)` pour chaque ligne de `Y`.
`X[Y,fun(V),by=.EACHI]` `[V1!=0]`	Agréger, calculer `fun(V)` pour chaque ligne de `Y`, et extraire les résultats $\neq 0$.
`X[Y,mult="first"]` `X[Y,mult="last"]`	Agréger et afficher la première/dernière ligne de chaque groupe.

$\hookrightarrow$ Et d'autres exemples dans l'aide du package `data.table` !

Les objets - Les Dates

Définir des formats

%d jour (en nombre), %a jour abrégé, %A jour non abrégé, %m mois (de 00 à 12), %b mois abrégé, %B mois non abrégé, %y année (deux chiffres), %Y année (4 chiffres).

Créer des Dates à partir de vecteurs de caractères

```
x=c("2015-01-01","2015-02-01","2015-03-01")
y=c("01/01/2015","01/02/2015","01/03/2015")
z=c("2015-01-01 00:00:01","2015-02-01 00:00:01")
```

Transformer `x` en Date : `x.date=as.Date(x)`
Transformer `y` en Date : `y.date=as.Date(y,format="%d/%m/%Y")`
Transformer `z` en Date/Heure au format POSIXlt : `z.date=strptime (z,"%Y-%m-%d %H:%M:%S")`
Créer une suite de dates : `seq(as.Date("2014-01-01"), as.Date("2015-01-01"), by = "month")`

Extraire des dates particulières

Date présente : `today=Sys.Date()`
Date, heure présentes : `date()`
Date, heure présentes POSIXlt : `as.POSIXlt(Sys.time(),"GMT")`
Date, heure présentes POSIXct : `as.POSIXct(Sys.time(),"UTC")`

Manipuler des dates

`format(today,"%B %d %Y")`	Modifier le format d'une date.
`sort(x.date, decreasing=TRUE)`	Ranger des dates par ordre décroissant.
`strftime(x.date,"%Y")`	Extraire les années.
`strftime(x.date,"%m")`	Extraire les mois.
`strftime(x.date,"%d")`	Extraire les jours.
`difftime()`	Calculer le temps séparant deux dates.

Programme

4. Importation et exportation de données
   - Importation
   - Exportation

Importation et exportation de données - Importation

Importer un fichier texte : la fonction `read.table()`

`read.table("chemin/donnees.txt",options)` crée un data.frame composé des données contenues dans `donnees.txt`.

`chemin` peut être un chemin classique ou une URL.

Options principales par défaut :

```
header=FALSE,sep="",quote="\"'",dec=".",row.names,
col.names,na.strings="NA"
```

Sortie : data.frame dont les composantes sont nommées `V1`, `V2`...

Modification des options par défaut (exemple) :

```
read.table("chemin/donnees.txt",header=T,sep="\t",
quote="\"",dec=",")
```

Pour les fichiers `.csv` : `read.csv()`, `read.csv2()` sont des variantes de `read.table` avec des options par défaut différentes.

Problèmes dûs au fichier possibles :

- Séparateur de colonnes ou décimal mal spécifié,
- tabulation remplaçant un espace,
- guillemet mal fermé ou apostrophe...

Une fonction de secours : `scan()`

`scan("chemin/donnees.txt",options)` crée un vecteur dont les composantes sont les éléments contenus dans `donnees.txt`.
Contrairement à `read.table`, elle permet de spécifier le mode des objets : `scan("chemin/donnees.txt",what=list("",0,0))` par exemple.

Importation rapide avec `data.table`

`data.t=fread("chemin/donnees.txt",options)` crée un data.table composé des données contenues dans `donnees.txt`, de façon rapide et automatique !

Autres packages utiles

- `Rcmdr` permet d'importer des données directement depuis le système.
- Le package `readxl` possède une fonction `read_excel` qui permet de lire directement des fichiers `.xls` ou `.xlsx`.
- `ROpenOffice` permet de gérer les fichiers OpenOffice (LibreOffice).
- `foreign` permet le transfert vers R de données créées à partir de SAS, S-PLUS, SPSS, STATA, MINITAB.
- `RMySQL` assure l'interface entre R et des systèmes de gestion de bases de données relationnels...

Importation et exportation de données - Exportation

La fonction `write.table()`

`write.table(x,"chemin/sorties.txt",options)` permet d'exporter dans un fichier `sorties.txt` un objet `x`, généralement un data.frame.

Options par défaut :

```
append=FALSE,quote=TRUE,sep="",eol="\n",na="NA",dec=".",
row.names=TRUE,col.names=TRUE,qmethod=c("escape","double")
```

Autres fonctions utiles

- `save(x,"chemin/sorties.txt",options)`
- `write(x,"chemin/sorties.txt",options)`
- `write.infile(x,"chemin/sorties.txt",options)` du package `FactoMineR`
- `write.csv(x,"chemin/sorties.csv",options)`

Programme

5. Les graphiques
   - Fenêtres graphiques
   - Commandes graphiques primaires
   - Personnalisation
   - Commandes graphiques secondaires
   - Exportation

Les graphiques - Fenêtres graphiques

Gérer les fenêtres graphiques

Les graphiques sont tracés dans le dispositif graphique actif.

`dev.new()`	Ouvrir une fenêtre graphique.
`dev.list()`	Afficher la liste des dispositifs graphiques ouverts.
`dev.cur()`	Connaître le dispositif graphique actif.
`dev.set(i)`	Rendre le ième dispositif actif.
`dev.off()`	Fermer le dispositif actif.
`dev.off(i)`	Fermer le ième dispositif.
`graphics.off()`	Fermer tous les dispositifs.

Partitionner la fenêtre graphique active

`par(mfrow=c(1,2))`	Partitionner la fenêtre graphique active en 2
`par(mfrow=c(2,1))`	sous-fenêtres de mêmes dimensions l'une à côté ou l'une au-dessous de l'autre.
`par(mfrow=c(2,3))`	Partitionner la fenêtre graphique active en 6 sous-fenêtres de mêmes dimensions (2×3).
`split.screen(c(2,1))`	Partitionner la fenêtre graphique active en 2 sous-fenêtres de mêmes dimensions l'une au-dessous de l'autre.
`split.screen(c(1,3),`	Partitionner la sous-fenêtre 2 en 3 fenêtres de
`screen = 2)`	mêmes dimensions les unes à côté des autres.
`screen(2)`	Rendre la sous-fenêtre 2 active.
`erase.screen(2)`	Effacer la sous-fenêtre 2.
`close.screen(all=T)`	Sortir du mode `split.screen`.

Partitionner la fenêtre graphique active avec `layout`

`layout(m,widths=w,heights=h)` permet de partitionner la fenêtre active en sous-fenêtres pas nécessairement de mêmes dimensions.

Arguments : `m` matrice de taille nl×nc d'entiers naturels, `w` et `h` vecteurs de réels positifs ou de type `lcm(x)`.

La fenêtre graphique est découpée en nl lignes et nc colonnes, créant ainsi nl×nc carreaux unitaires.
Les éléments de `m` indiquent les numéros des graphiques qui doivent être dessinés dans chaque carreau unitaire.
Le vecteur `w` indique les largeurs relatives ou en cm des colonnes.
Le vecteur `h` indique les hauteurs relatives ou en cm des lignes.

Pour visualiser une fenêtre graphique partitionnée avec `layout`, on utilise `layout.show(n)` où n est le nombre de sous-fenêtres.

Les graphiques - Commandes graphiques primaires

Des exemples : `demo(graphics)`

La fonction `plot`

`x` et `y` vecteurs numériques, `z1` et `z2` facteurs, tous de même longueur n, `donnees` data.frame de composantes `x,y,z1,z2`.

`plot(x,options)`	Tracer les valeurs de `x` en fn de (1,...,n).
`plot(z1,options)`	Tracer un diagramme en tuyaux d'orgue de `z1`.
`plot(x,y)`/`plot(y~x)`/ `plot(y~x,data=donnees)`	Tracer les valeurs de `y` en fn de celles de `x`.
`plot(x~z1,data=donnees)`	Tracer des boîtes à moustaches des valeurs de `x` en fn des niveaux de `z1`.
`plot(z1~z2,data=donnees)`	Tracer un diagramme en bandes des valeurs de `z1` en fn de `z2`.
`plot(donnees)`	Donner un scatterplot de `donnees`.
`plot(sin,0,5)`	Tracer le graphe de sinus sur $[0, 5]$.

Autres fonctions utiles

`curve(x^3,0,5)`	Tracer le graphe de la fonction $x \mapsto x^3$ sur $[0, 5]$.
`curve(sin,0,5)`	Tracer le graphe de la fonction sinus sur $[0, 5]$.
`hist(x,freq=TRUE)`	Représenter `x` sous forme d'histogramme.
`hist(x,freq=FALSE)`	Représenter `x` sous forme d'histogramme d'aire 1.
`hist(x,prob=TRUE)`	Idem.
`boxplot(x)`	Représenter `x` sous forme de diagramme en boîte.
`boxplot(x~z1)`	Représenter `x` sous forme de diagramme en boîte selon les niveaux de `z1`.
`pie(table(z1))`	Représenter `z1` sous forme de diagramme circulaire.
`barplot(matrix (table(z1)))`	Représenter `z1` sous forme de diagramme en bande.
`scatterplot(y~x)` (package car)	Tracer un scatterplot de `y` en fonction de `x`.
`pairs(donnees)`	Tracer les graphes bivariés entre composantes.
`qqnorm(x)`	Tracer les quantiles de `x` en fn de ceux attendus d'une loi normale.
`qqplot(x,y)`	Tracer les quantiles de `y` en fn de ceux de `x`.

Le package `lattice`

`xyplot(y~x)`	Tracer `y` en fn de `x`.
`xyplot(y~x|z1)`	Tracer `y` en fn de `x` selon les niveaux de `z1`.
`xyplot(y~x|z1*z2)`	Tracer `y` en fn de `x` selon les niveaux croisés de `z1` et `z2`.

Les graphes en 3D

`z` un vecteur numérique

`contour(x,y,z)`	Tracer des courbes de niveau.
`filled.contour (x,y,z)`	Tracer des courbes de niveau en couleurs.
`image(x,y,z)`	Représenter une grille de rectangles, dont les couleurs représentent les valeurs de `z`.
`persp(x,y,z)`	Représenter les valeurs de `z` en perspective.

Les graphiques - Personnalisation

Les modes de personnalisation

On peut personnaliser un graphique de trois façons différentes :

- en spécifiant les options directement en argument dans la fonction graphique utilisée (temporaire),
- en spécifiant les options au préalable à l'aide de `par(options)` (permanent),
- à l'aide de commandes graphiques secondaires.

Attention, toutes les options de personnalisation ne sont pas disponibles en argument dans les fonctions graphiques !

Les options de personnalisation

Quelques options spécifiques à `plot` :

`type= "l","p"...`	Type de tracé.
`main="titre"`	Titre du graphe.
`sub="sstitre"`	Sous-titre du graphe.
`xlim=c(0,5),ylim=c(0,2)`	Limites des axes.
`xlab="abs",ylab="ord"`	Annotations des axes.

Quelques options "génériques" (liste donnée par `help(par)`)

`col="red" col.axis`...	Couleurs du graphe, des axes... `colors()` donne la liste des couleurs par défaut.
`bty="n"` ...	Tracé du cadre.
`mar=c(0,0,0,0)`	Marges.
`pch="+","o","."`...	Symbole utilisé pour les points.
`cex=1.5,cex.axis,cex.lab, cex.main`...	Taille des symboles ou caractères.
`font=1, font.axis`...	Police du texte.
`lty="1", lwd=1.5`	Type et largeur des lignes tracées.

Les graphiques - Commandes graphiques secondaires

Ajout d'éléments à un graphe

`points(x,y)`	Ajouter le graphe des valeurs de `y` en fonction de celles de `x` avec des points.
`lines(x,y)`	Idem à `points` mais en trait continu.
`abline(a,b)`	Ajouter une droite d'ordonnée à l'origine `a`, de pente `b`.
`abline(h=o)`	Ajouter une droite horizontale en l'ordonnée `o`.
`abline(v=a)`	Ajouter une droite verticale en l'abscisse `a`.
`segments(x0,y0, x1,y1)`	Tracer des segments joignant les points `(x0,y0)` aux points `(x1,y1)`.
`arrows(x0,y0,x1, y1,angle=30, code=1,2,3)`	Tracer des flèches dont la pointe forme un angle de 30 avec l'axe aux points `(x1,y1)`, `(x0,y0)` ou aux deux.
`rect(x1,y1,x2,y2)`	Tracer un rectangle.
`polygon(x,y)`	Tracer un polygone reliant les points de coordonnées `(x,y)`.

Ajout d'éléments à un graphe (suite)

`text(abs,ord,"bla")`	Ajouter le texte `bla` en `(abs,ord)`.
`text(abs,ord, expression())`	Ajouter une expression mathématique en `(abs,ord)`.
`text(abs,ord, as.expression (substitute()))`	Ajouter une expression mathématique incluant la valeur d'une variable numérique en `(abs,ord)`.
`mtext(texte,side=3)`	Ajouter `texte` dans une des marges.
`legend(x,y)`	Ajouter la légende en (x,y).
`title(main,sub)`	Ajouter un titre, un sous-titre.
`axis(side,vect)`	Ajouter un axe ou des graduations sur les axes.
`rug(x)`	Ajouter les valeurs de `x` sur l'axe des abscisses.
`locator(n,type="n")`	Retourner les coordonnées (x,y) ou ajouter des symboles aux points de coordonnées (x,y) après n clics sur le graphe.
`text(locator(1),texte)`	Utiliser la souris pour placer du texte.
`identify(x,y,texte)`	Mettre en évidence des points du graphe en cliquant dessus, leur accoler du texte.

Les graphiques - Exportation

Exportation directe à partir de l'outil `Plots`

Exportation depuis la console

1. Ouvrir un fichier graphique à l'aide de `postscript("chemin/mesgraphiques.eps")` ou `pdf("chemin/mesgraphiques.pdf")`,
2. Tracer les graphiques à exporter,
3. Fermer la fenêtre à l'aide de `dev.off()`.

Programme

6. Les fonctions et la programmation
   - Les fonctions
   - Les boucles et structures de contrôle

Ecrire ses propres fonctions

▶ Créer une fonction avec les arguments `x` et `y` en entrée et retournant en sortie `res1` ou/et `res2` renommés `nom1`, `nom2` :

```
mafonction=function(x,y){instructions; return(nom1=res1)}
mafonction=function(x,y){instructions;
return(list(nom1=res1,nom2=res2))}
```

Pour exécuter cette fonction pour `x=3` et `y=1` : `mafonction(3,1)`.

▶ Créer un nouvel opérateur binaire :

`"%!%"=function(x,y){instructions}`, utilisé avec `x%!%y`.

Quelques commandes utiles

`print(x)`	Afficher l'objet `x`.
`assign(x,pos=1)`	Assigner l'objet `x` dans le répertoire en position 1.
`expression=parse(texte)`	Transformer `texte` en expression (non évaluée).
`eval(expression)`	Evaluer `expression`.
`deparse(expression)`	Transformer `expression` en texte.
`substitute(expression)`	Substituer dans une expression les valeurs des variables.
`stop("message")`	Stopper l'exécution en affichant `message`.
`warning("message")`	Afficher le message d'avertissement `message`.

Un conseil : découper les programmes trop longs en plusieurs fonctions, qui pourront être testées individuellement.

Les fonctions et la programmation - Les boucles et structures de contrôle

Les boucles `for`, `while` et `repeat`

`for (i in vecteur) {instructions}`
↪ Exécute `instructions` pour chaque élément de `vecteur`.

`while (condition) {instructions}`
↪ Exécute `instructions` tant que `condition` est `TRUE`.

`repeat {instructions;if (condition) break}`
↪ Exécute `instructions` jusqu'à ce que `condition` soit `TRUE`.

Un conseil : Remplacer au maximum l'utilisation des boucles par une vectorisation des objets manipulés (data.table !).

Les structures de contrôle `if` et `else`

`if (condition) {instructions} else {alternatives}`
↪ Exécute `instructions` si `condition` est `TRUE`, `alternatives` sinon.

Programme

7. Analyse statistique avec R
   - Statistiques descriptives
   - Tests et intervalles de confiance
   - Modèles, formules et contrastes
   - Analyses de modèles

Analyse statistique avec R - Statistiques descriptives

Les fonctions vues précédemment permettent une première analyse descriptive.

Pour aller plus loin...

- Analyse en Composantes Principales : `princomp`, `PCA` du package `FactoMineR`.
- Analyse Factorielle des Correspondances : `CA` du package `FactoMineR`.
- Analyse des Correspondances Multiples : `MCA`, `dimdesc` du package `FactoMineR`.
- Classification Ascendante Hiérarchique : `agnes` du package `cluster`, `catdes` du package `FactoMineR`.
- K-Means : `kmeans`, `catdes` du package `FactoMineR`.

Analyse statistique avec R - Tests et intervalles de confiance

Tests dans des modèles paramétriques simples

- Test de Student sur l'espérance d'un échantillon gaussien, test de Student ou Welch de comparaison d'espérances de deux échantillons gaussiens : `t.test`.
- Test de Fisher-Snedecor de comparaison de variances d'échantillons gaussiens : `var.test`.
- Tests asymptotiques sur une probabilité ou de comparaison de probabilités : `prop.test`.
- Test exact sur une probabilité : `binom.test`.
- Test exact de Fisher d'égalité de probabilités : `fisher.test`.

Evaluation de la puissance de certains de ces tests : `pwr.t.test`...

Tests non paramétriques

- Tests du khi-deux : `chisq.test`.
- Test exact de Fisher ou Fisher-Freeman-Halton : `fisher.test`.
- Test de Kolmogorov-Smirnov : `ks.test`.
- Tests de normalité de Lilliefors et de Shapiro-Wilk : `lillie.test` (package `nortest`), `shapiro.test`.
- Test de Mann-Whitney-Wilcoxon : `wilcox.test`.

Intervalles de confiance et bootstrap

Certains intervalles de confiance dans des modèles paramétriques simples sont obtenus directement à partir des fonctions de tests.

Pour construire des intervalles de confiance non programmés dans R à partir d'une fonction pivotale, on peut utiliser les fonctions `qloi` donnant les quantiles des lois usuelles, ou bien des méthodes de Monte Carlo et la fonction `quantile` permettant d'évaluer des quantiles empiriques.

La fonction `boot.ci` du package `boot` permet de construire des intervalles de confiance bootstrap.

Analyse statistique avec R - Modèles, formules, contrastes

Une formule est un objet permettant de décrire un modèle de type régression ou discrimination.
L'opérateur $\sim$ sépare la variable à expliquer `y` (vecteur ou matrice) des variables explicatives `x1,x2,x3`...

Formules de base

`y~.`	Modèle complet.
`y~1`	Modèle dont la seule variable explicative est la constante.
`y~x1+x2`	Modèle dont les variables explicatives sont la constante, `x1` et `x2`.
`y~-1+x1+x2` ou `y~0+x1+x2`	Modèle dont les variables explicatives sont `x1` et `x2`.

Formules avec interactions

`y~x1+x2+x1:x2` ou `y~x1*x2`	Modèle avec la constante, `x1`,`x2` et `x1x2`.
`y~(x1+x2+x3)^2`	Modèle avec la constante et les interactions entre `x1`,`x2`,`x3` jusqu'à l'ordre 2.
`y~x1+x2%in%x1`	Précise les hiérarchies. Idem à `y~x1+x1:x2`.
`y~(x1+x2+x3)^2-x1:x3`	Modèle `y~(x1+x2+x3)^2` sans `x1x3`.
`y~poly(x1,degree=2)`	Modèle avec des termes polynômiaux en `x1` jusqu'au degré 2.
`y~ log(x1)`	Modèle avec la constante et `log(x1)`.
`y~x1>1`	Modèle avec la variable prenant la valeur 1 si `x1>1`, 0 sinon.
`I(x1^2)`	Permet de suspendre l'interprétation de ^ comme opérateur formule et de revenir à sa signification arithmétique.

Facteurs et contrastes

Pour éviter la surparamétrisation des modèles d'analyse de la variance, on impose une contrainte (un contraste) aux facteurs.
Soit un facteur `fact` à I modalités, α_i le coefficient du modèle associé au $i$ème niveau du facteur `fact`.

`contrasts(fact)`	Voir la valeur du contraste de `fact`.
`contrasts(fact)=` `contr.sum(levels(fact))`	Affecter un contraste de type "somme" $\sum_{i=1}^{I} \alpha_i = 0$ à `fact`.
`contrasts(fact)=contr.` `treatment(levels(fact))`	Affecter un contraste de type "traitement-témoin" $\alpha_1 = 0$.
`contrasts(fact)=contr.` `poly(levels(fact))`	Affecter un contraste polynômial.

On peut aussi changer un contraste ponctuellement en option :
`objet=lm(y~fact1+fact2,contrasts=list(fact1="contr.sum", fact2="contr.poly"),data=donnees)`.

Analyse statistique avec R - Analyses de modèles

Fonctions pour l'analyse de modèles

- Régression linéaire multiple : `lm`, `anova` pour les tests, package `leaps`, `influence.measures`, `predict`.
- Analyse de la variance / covariance : `lm` puis `anova` pour les tests.
- Régressions logistique et linéaires généralisées : `glm`, `cv.glm`.
- Analyses discriminantes : `lda`, `qda` (package `MASS`).
- Discrimination par arbre de décision CART : `rpart` (`rpart`).
- Régression PLS : `mvr` (`pls`), `plsRglm` (`PlsRglm`),
- Régression LASSO, elastic net : `glmnet` (`glmnet`),
- Support Vector Machines : `svm` (`e1071`), `ksvm` (`kernlab`)
- Forêts aléatoires : `randomForest` (`randomForest`),
- Boosting : `gbm` (`gbm`),

et bien d'autres encore...

La sortie nommée par exemple `sortie.lm` de l'une de ces fonctions est une liste d'objets.

On peut afficher les noms de ces objets avec `attributes(sortie.lm)`.

Les objets de `sortie.lm` peuvent être extraits comme n'importe quels objets d'une liste : par exemple `sortie.lm$coefficients`, `sortie.lm$residuals`, `sortie.lm$fitted.values`...

`print(sortie.lm)` affiche le résultat de l'analyse menée,
`summary(sortie.lm)` donne un résumé de l'analyse,
`plot(sortie.lm)` trace des graphes relatifs à l'analyse.

↪ On peut consulter les aides spécifiques à l'aide de `help(summary.lm)`, `help(plot.lm)`...